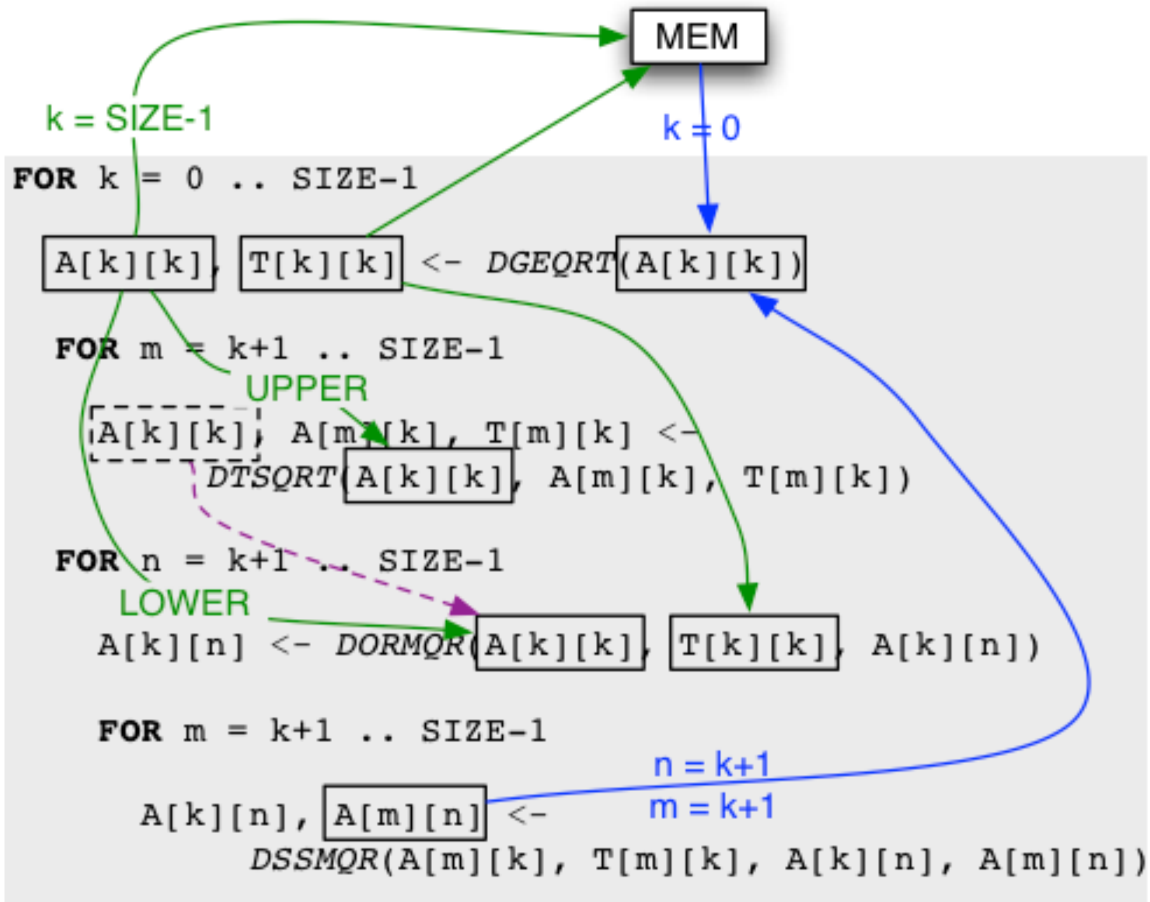
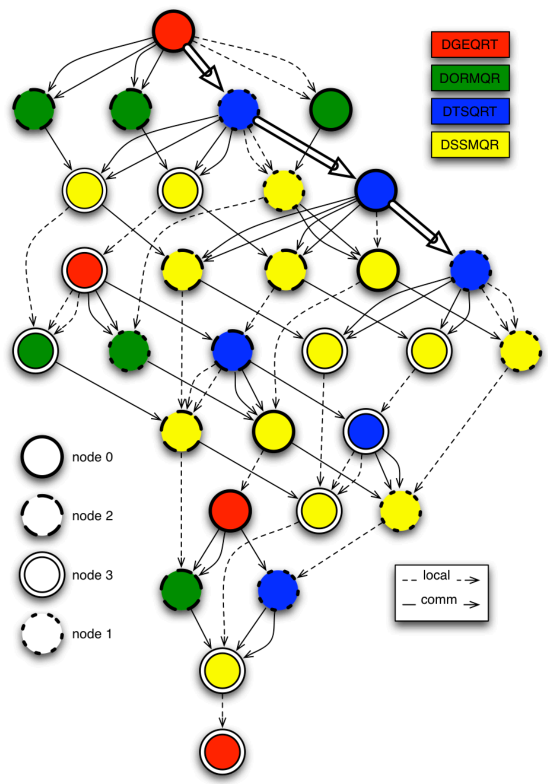
Efficiently Scheduling Task Dataflow Parallelism

Hans Vandierendonck
Queen’s University Belfast
EASC’15
Edinburgh, UK

Queen's University Belfast

Linear Algebra: QR Decomposition





Source: http://icl.cs.utk.edu/parsec/

Swan

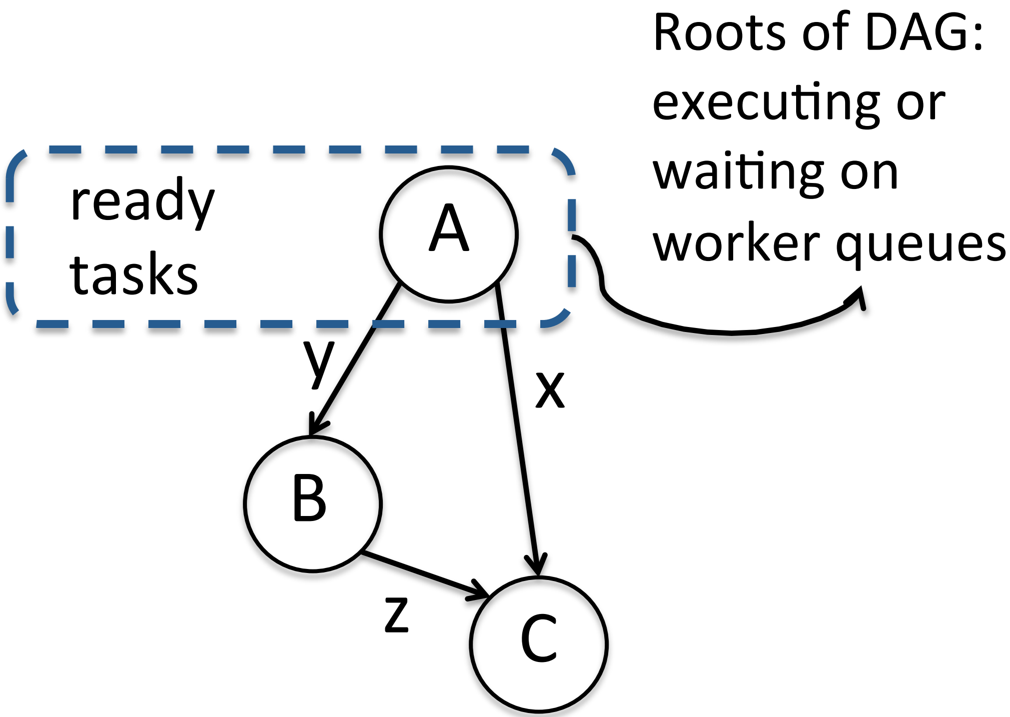
- Task dataflow model established in HPC
  - SuperMatrix, QUARK, StarSS/OmpSS, StarPU, Xkaapi, …
  - Supports runtime optimization:
    - Locality-aware scheduling, scheduling in distributed memory systems, runtime-aware coherence protocols, …
- Application domain is not restricted to HPC
  - Pipeline parallelism [HotPar'11, PACT'11]
  - Fine-grain queues for PARSEC benchmarks [SC'14]
  - Exploring big data setting (EU-FP7 ASAP)
- Swan is designed not to be restricted to HPC

Swan = Cilk + Dataflow

- "Master" spawns tasks in program order
- Annotations of arguments indicate usage of argument
  - All side effects must be captured

```
void master() {
    data_type x, y, z;

    spawn A( in x, out y );
    spawn B( in y, out z );
    spawn C( inout x, in z );
    sync;
}
```



Swan Language Definition as a C++ Extension

- Versioned objects
  versioned<T> obj; // renaming
  unversioned<T> obj; // no renaming

- Argument annotations
  indep<T> *read-only*
  outdep<T> *read/write but no exposed reads*
  inoutdep<T> *read/write*
  cinoutdep<T> *commutative*
  reduction<M> *reduction*
  -- T is a C++ type
  -- M is a C++ structure describing the monoid with type T, an identity value and a reduction operator

- Independent fork/join
  int x;
  spawn f(x);
  …
  sync;

- Dependency-aware fork/join
  versioned<int> x;
  spawn f((indep<int>)x);
  …
  sync;

- Retain implicit sync at end of procedure

Queen's University Belfast

From Side Effects to Dependences

		First Task				
Second Task		input	output	in/out	commutative in/out	reduction
	input	none	true	true	true	true
	output	anti	output	output	output	output
	in/out	anti	true	true	true	true
	commutative in/out	anti	true	true	none$^{(x)}$	true
	reduction	anti	true	true	true	none$^{(r)}$

Assumption: Two tasks access the same variable with annotations as in table

Notes:

- none$^{(x)}$: no dependence, except for enforcement of mutual exclusion
- none$^{(r)}$: no dependence, except for privatization of variables and final reduction
- Renaming (new copy of variable), applied only on output annotation

Queen's University Belfast

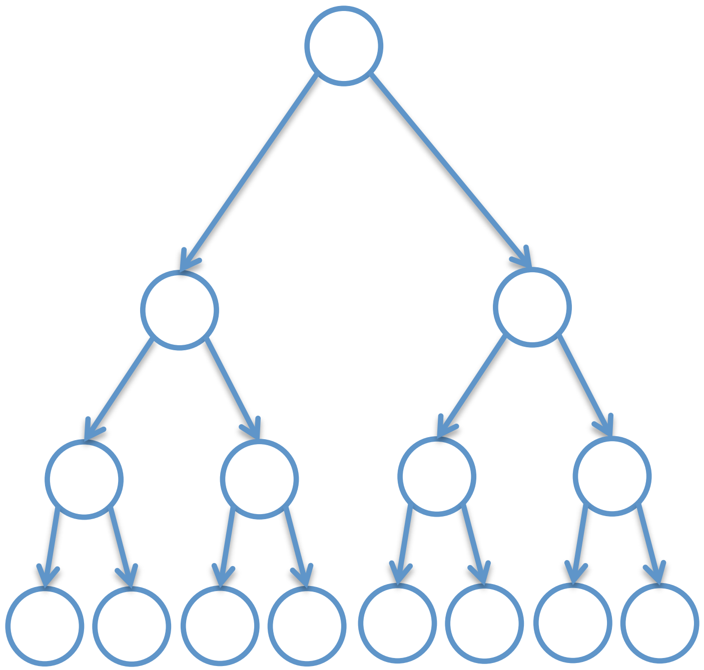
Linear Algebra: QR Decomposition

```
typedef versioned<float[]> block_t;
typedef indep<float[]> bin;
typedef outdep<float[]> bout;
typedef inoutdep<float[]> binout;

// Initialise matrices using blocked matrix representation
versioned<float[]> A[N][N]; A[i][j] = new versioned<float[]>(D*D); …;
versioned<float[]> T[N][N]; …;
// QR dedomposition: A contains Q and R; T is temporary storage
for( k=0; k < N; ++k ) {
        spawn dgeqrt((binout)A[k][k], (bout)T[k][k]);
        for( m=k+1; k < N; ++k ) {
                spawn dtsqrt( (binout)A[k][k], (binout)A[m][k], (binout)T[m][k] );
        for( n=k+1; n < N; ++n )
                spawn dormqr((bin)A[k][k], (bin)T[k][k], (binout)A[k][n] );
                for( m=k+1; m < N; ++m )
                        spawn dssmqr( (bin)A[m][k], (bin)T[m][k], (binout)A[k][n], (binout)A[m][n] );
}
sync;
```

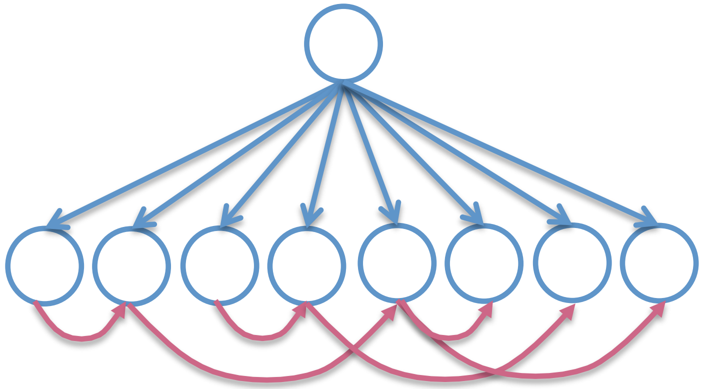
Unified Scheduler

Typical Cilk spawn tree



- Deep spawn tree

Typical task dataflow spawn tree

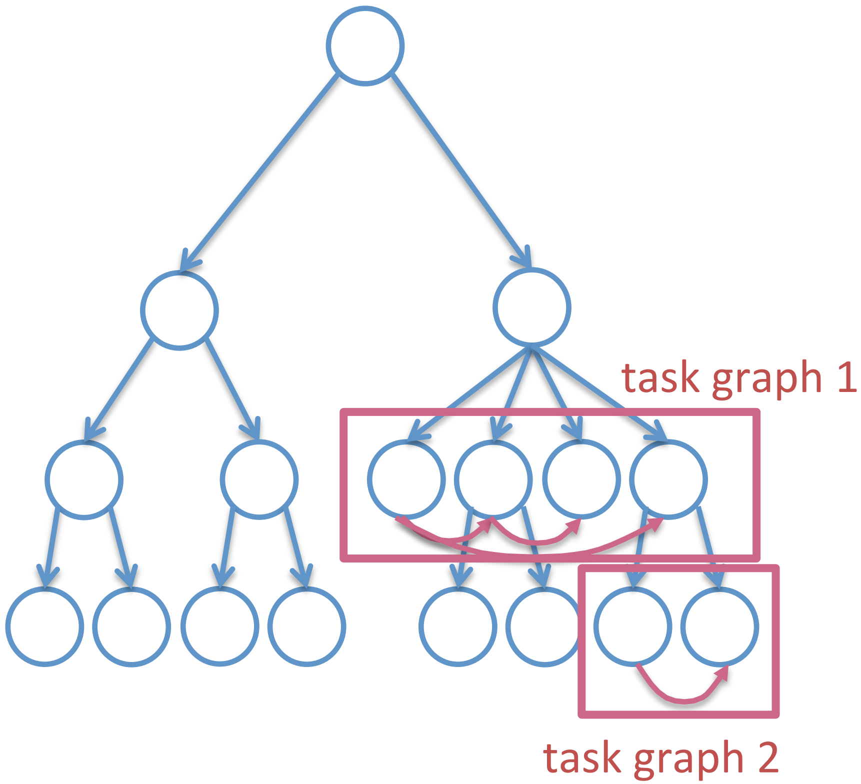


- Shallow spawn tree
- Dataflow dependencies between children
- Every task in the spawn tree may organize its children in a dataflow graph
- Arbitrary nesting of fork/join and task graphs

Queen's University Belfast

Unified Scheduler

Mixed fork/join – dataflow spawn tree



Qualitative properties

- Cannot maintain busy-leaves principle
  - Non-ready tasks are non-busy leaves
- Maintains work-first principle
  - Execute task immediately if data dependencies allow it
  - Keeps the task graph small
- Work stealing in dataflow graphs more frequent then in fork/join

Queen's University Belfast

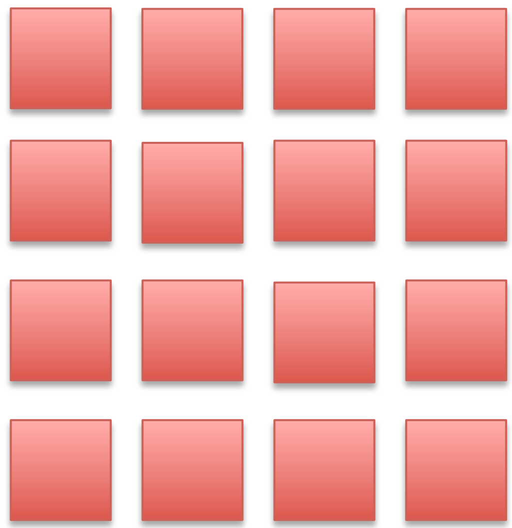
Work Stealing

- Extend data structures scheduler
  - Cilk's spawn dequeue + one ready list per worker thread
- If worker's ready list is not empty
  - Select and execute a task from the worker's ready list
- Random work stealing
  - Select a random victim worker thread
  - If victim's ready list not empty, steal half of ready tasks [Hendler & Shavit, '02]
  - Place 1 stolen task on spawn deque and execute
- Unconditional steal, "provably-good" steal
  - As in Cilk: continue with parent if possible, else do random work stealing algorithm

Queen's University Belfast

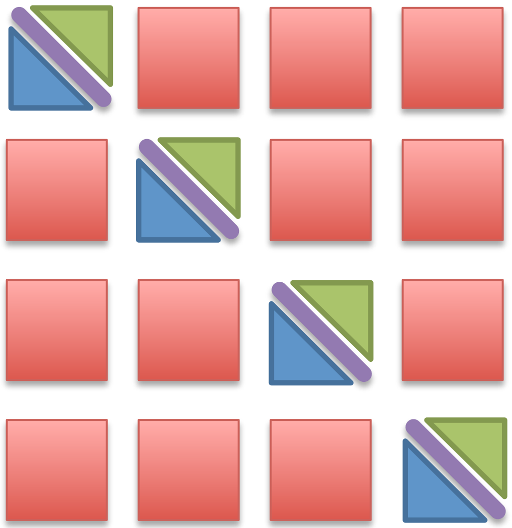
PLASMA/QUARK Setup (i)

- Matrix represented by blocks
- Parallelize L3 BLAS operations as "algorithms-by-blocks"
  - DAG of by-block operations
  - Performance independent of algorithm variation [Chan *et al*, PPoPP'08]



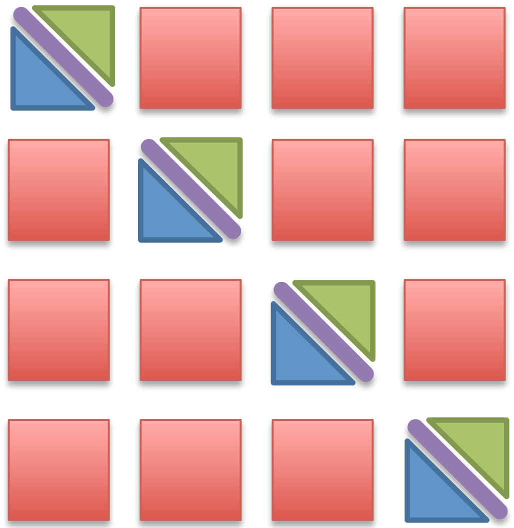
PLASMA/QUARK Setup (ii)

- Matrix represented by blocks
- Some operations touch only part of a block
  - Upper/lower triangles, diagonal
  - Precise dependence tracking increases parallelism



Swan Interface to PLASMA

- swan_desc<T>
  - Overlays a 2D array of Swan objects (unversioned<void>) over matrix
  - T is float or double
- BLAS wrappers
  - plasma_indep<T,part=lo|diag|up>
    plasma_outdep<T,part=lo|diag|up>
    plasma_inoutdep<T,part=lo|diag|up>
  - part is lo, diag or up, or a combination of these
- Parallel algorithms
  - desc.get_indep<part=...>(), etc.



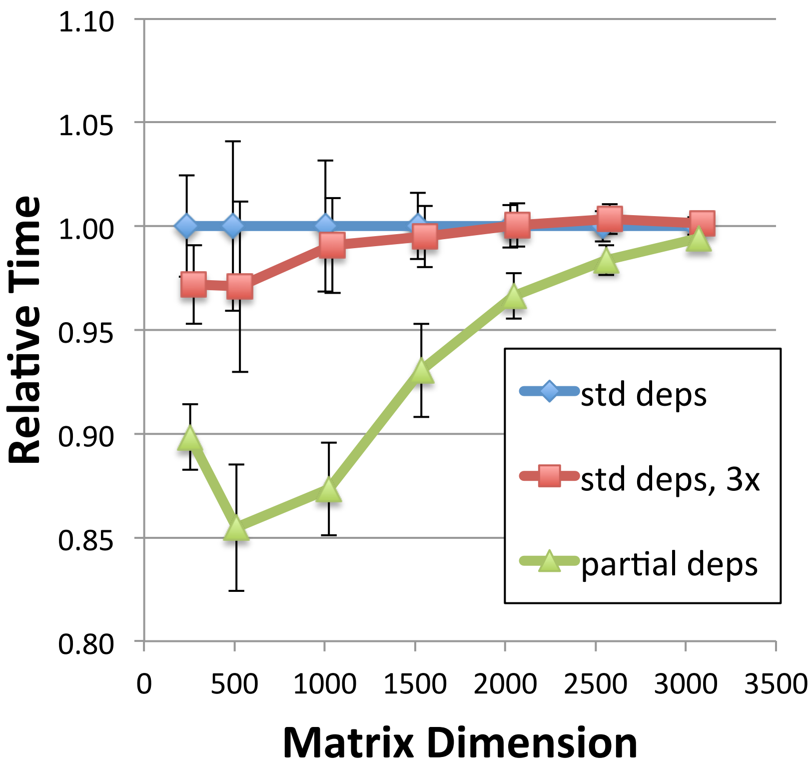
- Dependence tracking
  - Lo/diag/up sub-parts treated as independent variables, for all blocks
  - QUARK locates ready tasks by walking list of tasks waiting on a block

Queen's University Belfast

Experimental Evaluation

- Swan-based API for PLASMA routines
  - Drop-in replacement
  - Same parallelism exploited as QUARK
  - Call into same core BLAS routines
  - QUARK exploits memory locality (affinity)
- Evaluation environment
  - 2x 8-core Intel Xeon E5-2650 2GHz
  - PLASMA 2.6.0
  - CentOS 6.5, gcc 4.9.2
  - MKL version 11.1.2, single-threaded

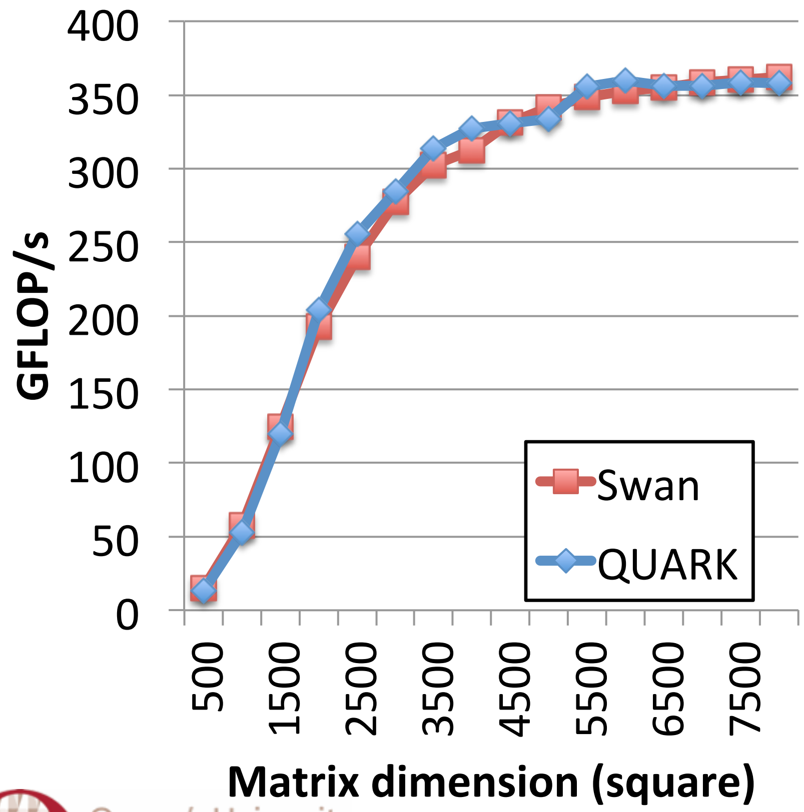
Quantifying Overhead



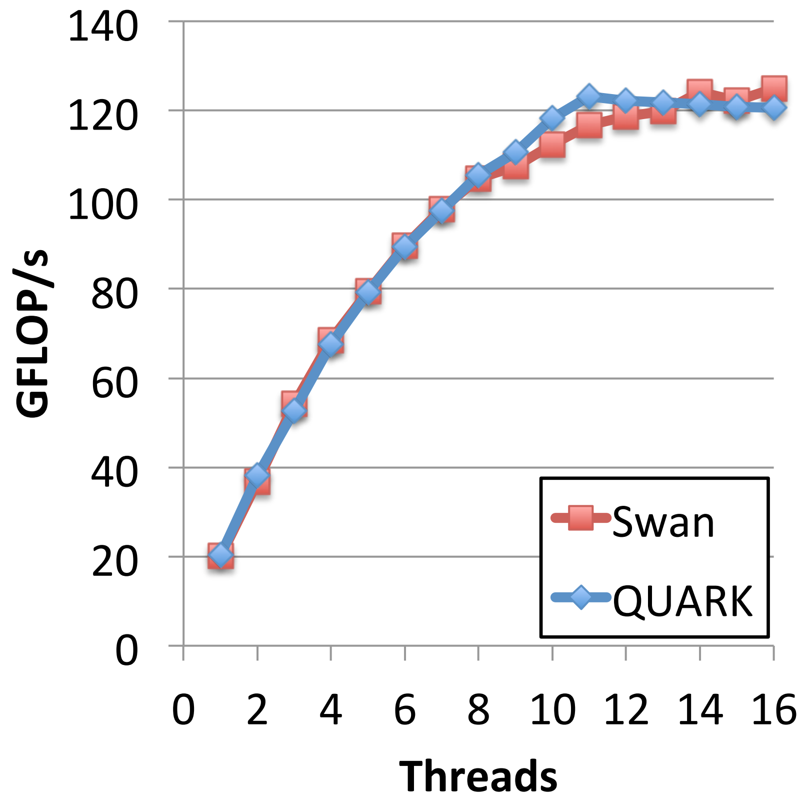
- Measure overhead of tracking dependences on 3 parts of an object
  - QR, 16 threads
  - Normalized to standard dependences on full matrix blocks
- Overhead statistically insignificant
- Benefit on small matrices
  - 2x2 to 16x16 matrix blocks
  - Increased parallelism

Cholesky (spotrf)

16 threads

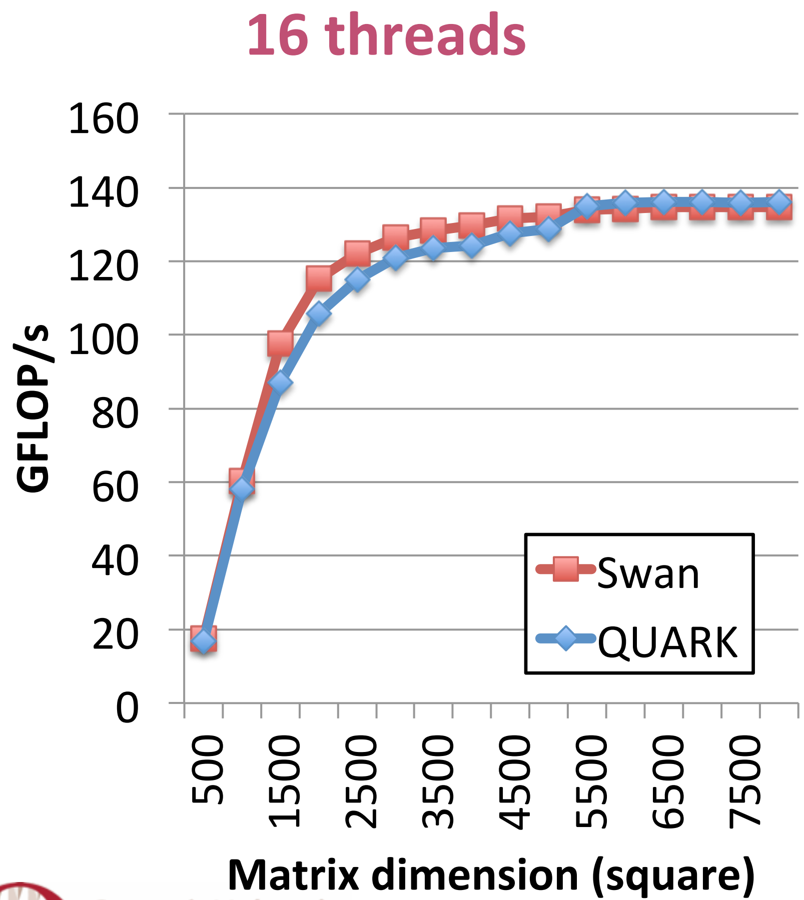


N=1500

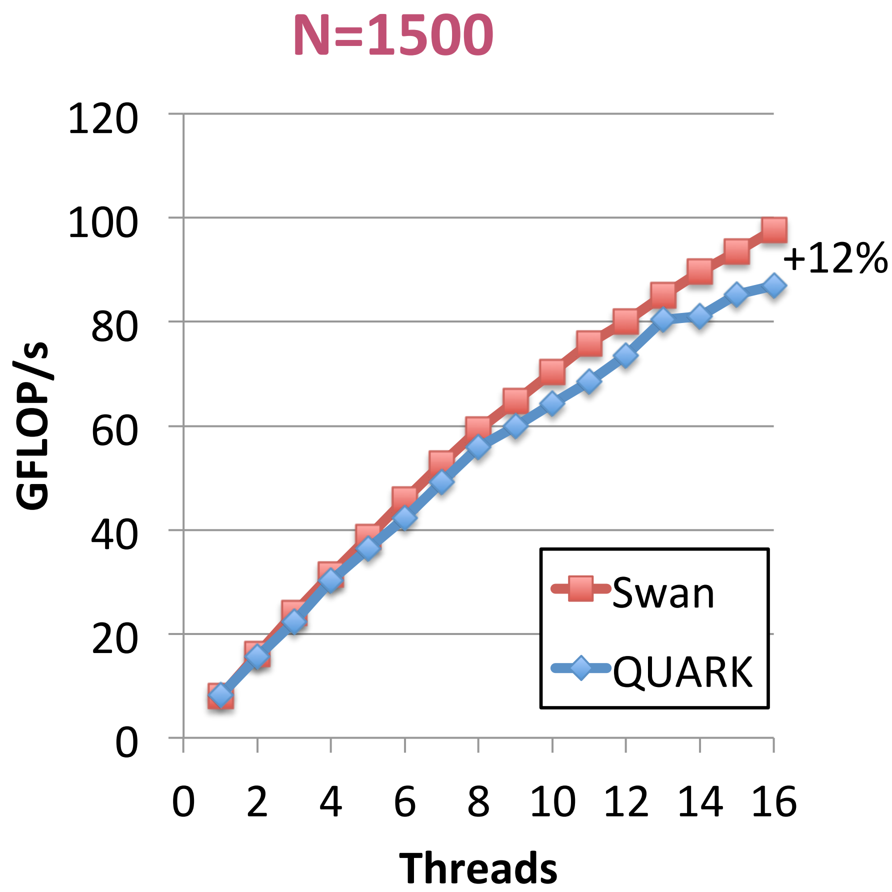


Queen's University Belfast

QR (dgeqrf)

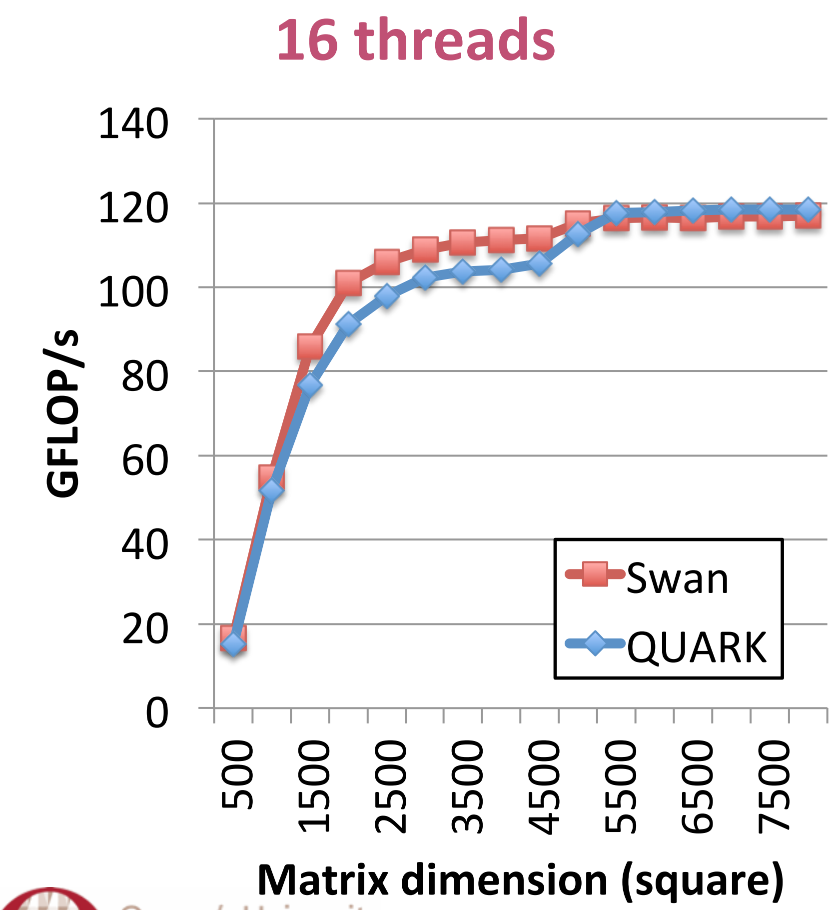


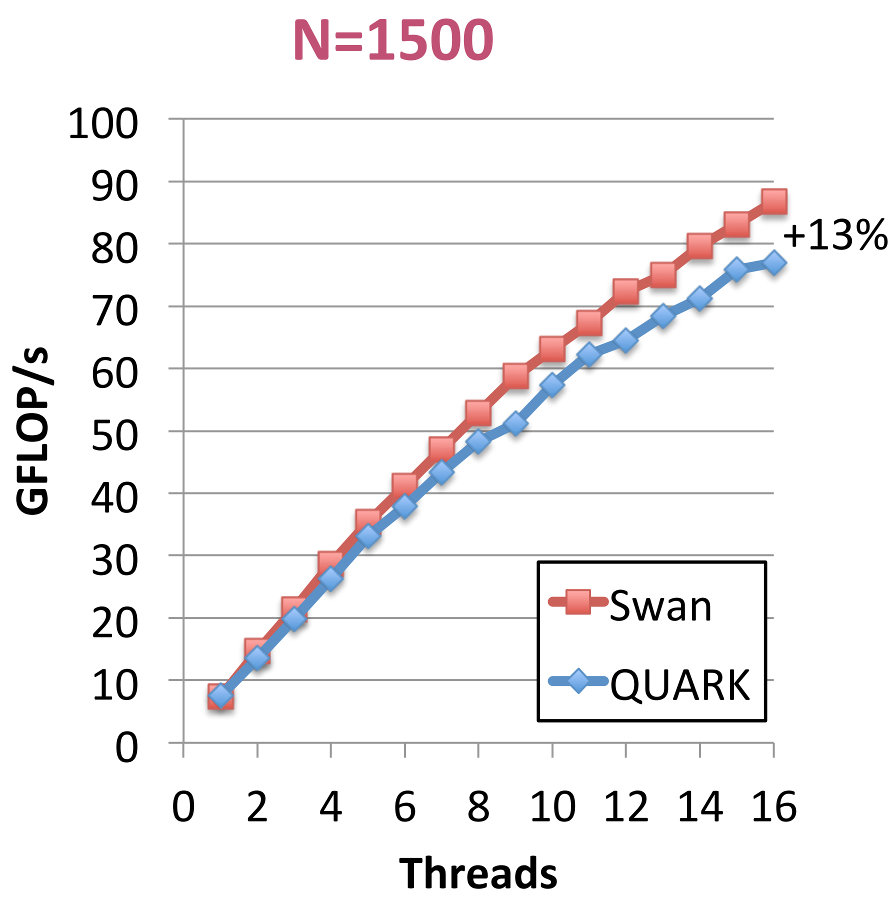
16 threads
160
140
120
100
80
60
40
20
0
GFLOP/s
500
1500
2500
3500
4500
5500
6500
7500
Matrix dimension (square)
Swan
QUARK



N=1500
120
100
80
60
40
20
0
GFLOP/s
0
2
4
6
8
10
12
14
16
Threads
+12%
Swan
QUARK

LU (dgetrf_incpiv)





Queen's University Belfast

Conclusion

- Swan is a language approach to task dataflow programming and execution
- Swan is a good alternative to QUARK for linear algebra kernels
  - Out-perform QUARK for a range of mid-size matrices
  - Performance difference grows with number of threads
- Future work:
  - Include memory locality / affinity in scheduling
  - Include NUMA awareness in work stealing

Queen's University Belfast

Questions?

- Collaborators:
  - Dimitris S. Nikolopoulos, George Tzenakis (QUB)
  - Polyvios Pratikakis (FORTH ICS)
  - Kallia Chronaki (BSC)
- Available for your experimentation:
  - http://github.com/hvdieren/swan/
  - http://github.com/hvdieren/parsec-swan/

SEVENTH FRAMEWORK PROGRAMME

EPSRC
Engineering and Physical Sciences Research Council

Queen's University Belfast

MARIE CURIE ACTIONS